[Sentiment Analysis] (CheatSheet)

1. Data Collection

- **Read data from CSV**: `df = pd.read_csv('file.csv')`
- **Read data from JSON**: `df = pd.read_json('file.json', lines=True)`
- **Load text data from files**: `text = open('file.txt', 'r').read()`
- **Fetch data via API**: `data = requests.get('API_ENDPOINT').json()`
- **Stream data from Twitter**: `tweets = tweepy.Cursor(api.search, q='keyword', lang="en").items()`
- **Web scraping with BeautifulSoup**: `soup = BeautifulSoup(page.content, 'html.parser')`
- **Use Pandas to read HTML tables**: `tables = pd.read_html('URL')`
- **Read Excel files**: `df = pd.read_excel('file.xlsx')`
- **Read data from SQL database**: `df = pd.read_sql_query("SELECT * FROM table_name", connection)`
- **Collect data from online forums (e.g., Reddit)**: `submissions = reddit.subreddit('subreddit').hot(limit=100)`

2. Data Preprocessing

- **Tokenization with NLTK**: `tokens = nltk.word_tokenize(text)`
- **Tokenization with spaCy**: `doc = nlp(text); tokens = [token.text for token in doc]`
- **Lowercasing**: `lower_text = text.lower()`
- **Removing punctuation**: `import string; text = text.translate(str.maketrans('', '', string.punctuation))`
- **Removing stopwords (NLTK)**: `filtered = [word for word in tokens if word not in stopwords.words('english')]`
- **Removing stopwords (spaCy)**: `filtered = [token.text for token in doc if not token.is_stop]`
- **Stemming (NLTK)**: `stemmed = [PorterStemmer().stem(word) for word in tokens]`
- **Lemmatization (NLTK)**: `lemmatized = [WordNetLemmatizer().lemmatize(word) for word in tokens]`
- **Lemmatization (spaCy)**: `lemmatized = [token.lemma_ for token in doc]`
- **Remove non-alphabetic characters**: `alpha_only = [word for word in tokens if word.isalpha()]`
- **Remove short words**: `long_words = [word for word in tokens if len(word) > 2]`

By: Waleed Mousa

- **Regex operations for text cleaning**: `clean_text = re.sub(r'\W+', ' ', text)`
- **Expanding contractions**: `expanded_text = contractions.fix(text)`
- **Removing HTML tags**: `clean_text = BeautifulSoup(raw_html, 'html.parser').get_text()`
- **Detecting and converting emojis**: `emoji.demojize(text)`
- **Splitting text into sentences**: `sentences = nltk.sent_tokenize(text)`
- **Using custom stopword list**: `custom_filtered = [word for word in tokens if word not in custom_stopwords]`
- **Advanced Tokenization**: Use spaCy or Hugging Face's tokenizer for advanced tokenization needs.
  - `spacy.load("en_core_web_sm").tokenizer(text)`
  - `from transformers import AutoTokenizer; tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased"); tokenizer(text)`
- **Custom Stopword Removal**: Tailor your stopword list to better fit the context of your data.
  - `my_stopwords = set(stopwords.words('english') + ['customWord1', 'customWord2']); filtered_words = [word for word in tokenized_text if word not in my_stopwords]`
- **Lemmatization with Context**: Utilize spaCy's lemmatization which considers part of speech.
  - `lemmatized = [token.lemma_ for token in spacy.load("en_core_web_sm")(text)]`
- **Synonym Replacement**: Enhance the robustness of your dataset by incorporating synonyms.
  - `from nltk.corpus import wordnet; synonyms = [syn.lemmas()[0].name() for syn in wordnet.synsets(word)]`
- **Phrase Detection and Modeling with Gensim**: Detect and model phrases (bigrams, trigrams) to capture multi-word expressions.
  - `from gensim.models.phrases import Phrases, Phraser; phrases = Phrases(sentences); bigram = Phraser(phrases); bigram[sentence]`

3. Feature Extraction

- **Count Vectorization (Scikit-learn)**: `vectorizer = CountVectorizer().fit_transform(corpus)`
- **TF-IDF Vectorization (Scikit-learn)**: `tfidf_vectorizer = TfidfVectorizer().fit_transform(corpus)`
- **Word Embeddings (Word2Vec)**: `model = Word2Vec(sentences)`
- **Document Embedding with Doc2Vec**: `model = Doc2Vec(documents)`
- **Bag of N-grams**: `ngram_vectorizer = CountVectorizer(ngram_range=(1, 2)).fit_transform(corpus)`

- **Character-level Vectorization**: `char_vectorizer = CountVectorizer(analyzer='char').fit_transform(corpus)`
- **Using pre-trained Word Embeddings (e.g., GloVe)**: `embeddings = {word: vec for word, vec in glove}`
- **TF-IDF with N-grams**: `tfidf_ngram_vectorizer = TfidfVectorizer(ngram_range=(1, 3)).fit_transform(corpus)`
- **POS Tagging for Feature Engineering**: `pos_tags = nltk.pos_tag(tokens)`
- **Sentiment Scores as Features (VADER)**: `sentiment_scores = SentimentIntensityAnalyzer().polarity_scores(text)`
- **Feature Selection using Chi-Square**: `chi2_features = SelectKBest(chi2, k=1000).fit_transform(X_tfidf, y)`
- **Embedding with BERT (Transformers)**: `model = BertModel.from_pretrained('bert-base-uncased')`

4. Sentiment Analysis Models

- **Naive Bayes Classifier**: `model = MultinomialNB().fit(X_train, y_train)`
- **Logistic Regression**: `model = LogisticRegression().fit(X_train, y_train)`
- **Support Vector Machine (SVM)**: `model = SVC().fit(X_train, y_train)`
- **Random Forest Classifier**: `model = RandomForestClassifier().fit(X_train, y_train)`
- **Gradient Boosting Machines**: `model = GradientBoostingClassifier().fit(X_train, y_train)`
- **Deep Learning Model (Keras/TensorFlow)**: `model = Sequential([...])`
- **Using Pre-trained Models (e.g., BERT for sentiment)**: `model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased')`
- **Fine-tuning Pre-trained Models**: `model.fit(X_train, y_train)`
- **LSTM Network for Sentiment Analysis**: `model = Sequential([LSTM(units), Dense(1, activation='sigmoid')])`
- **CNN for Text Classification**: `model = Sequential([Conv1D(filters), MaxPooling1D(), Flatten(), Dense(1, activation='sigmoid')])`
- **Transfer Learning with Hugging Face Transformers**: `model = AutoModelForSequenceClassification.from_pretrained('model_name')`
- **Ensemble Learning Methods**: `ensemble = VotingClassifier(estimators=[('lr', model1), ('rf', model2)], voting='hard')`
- **Custom Sentiment Analysis with spaCy**: `custom_sentiment = spacy.load('en_core_web_sm'); custom_sentiment.add_pipe('custom_component')`

5. Evaluation and Interpretation

- **Accuracy Score**: `accuracy = accuracy_score(y_test, y_pred)`
- **Precision, Recall, and F1 Score**: `precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred)`
- **Confusion Matrix**: `conf_matrix = confusion_matrix(y_test, y_pred)`
- **ROC Curve and AUC**: `fpr, tpr, _ = roc_curve(y_test, y_scores); auc_score = auc(fpr, tpr)`
- **Classification Report**: `report = classification_report(y_test, y_pred)`
- **Cross-Validation Scores**: `cross_val_scores = cross_val_score(model, X, y, cv=5)`
- **Plotting Learning Curves**: `plot_learning_curve(model, 'Learning Curve', X, y)`
- **Mean Absolute Error (MAE) for regression models**: `mae = mean_absolute_error(y_true, y_pred)`
- **Mean Squared Error (MSE) for regression models**: `mse = mean_squared_error(y_true, y_pred)`
- **R-squared score for regression models**: `r2 = r2_score(y_true, y_pred)`
- **Kappa Score**: `kappa = cohen_kappa_score(y_true, y_pred)`
- **Log Loss**: `logloss = log_loss(y_true, y_pred_proba)`
- **Area Under the Precision-Recall Curve**: `auc_pr = average_precision_score(y_true, y_scores)`
- **Explaining Model Predictions (e.g., with SHAP)**: `explainer = shap.Explainer(model); shap_values = explainer(X)`
- **Feature Importance from models**: `importances = model.feature_importances_`

6. Visualization

- **Plotting Confusion Matrix**: `ConfusionMatrixDisplay.from_predictions(y_true, y_pred)`
- **ROC Curve Plotting**: `RocCurveDisplay.from_estimator(model, X_test, y_test)`
- **Precision-Recall Curve**: `PrecisionRecallDisplay.from_estimator(model, X_test, y_test)`
- **Feature Importance Visualization**: `sns.barplot(x=importances, y=feature_names)`
- **Word Cloud for Text Data**: `WordCloud(background_color='white').generate(' '.join(texts)).to_image()`
- **Histogram of Sentiment Scores**: `plt.hist(sentiment_scores, bins=50)`
- **Box Plot for Comparing Model Performances**: `sns.boxplot(data=models_performance)`
- **Heatmap of Confusion Matrix**: `sns.heatmap(conf_matrix, annot=True, fmt='d')`

- **Time Series Analysis of Sentiment Trends**: plt.plot(date_series, sentiment_series)
- **Distribution of Text Length or Sentiment**: sns.distplot(text_lengths); sns.distplot(sentiment_scores)

7. Advanced Techniques

- **Aspect-Based Sentiment Analysis**: Identify aspects and analyze sentiment for each aspect separately.
- **Sentiment Analysis with Transformers**: from transformers import pipeline; sentiment_pipeline = pipeline('sentiment-analysis'); results = sentiment_pipeline('We love Python!')
- **Multilingual Sentiment Analysis**: Support for multiple languages can be achieved with models like 'xlm-roberta-base'.
- **Sentiment Analysis in Streaming Data**: Use libraries like PySpark for real-time sentiment analysis on streaming data.
- **Using BERT for Fine-Tuning on Sentiment Data**: from transformers import BertTokenizer, TFBertForSequenceClassification; tokenizer = BertTokenizer.from_pretrained('bert-base-uncased'); model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased')
- **Leveraging GPT-3 for Sentiment Analysis**: OpenAI's GPT-3 can be fine-tuned or prompted for advanced sentiment analysis tasks.
- **Customizing VADER Sentiment Intensity Analyzer**: Customize the VADER lexicon to better fit your specific dataset and domain.
- **Sentiment Analysis with Deep Learning (RNN, LSTM, GRU)**: Building and training deep learning models like RNNs, LSTMs, and GRUs for capturing sequential dependencies in text data.
- **Zero-shot Sentiment Analysis**: Leverage pre-trained models to perform sentiment analysis without any training data on your specific task.
- **Transfer Learning in NLP**: Utilize a pre-trained NLP model and fine-tune it on your sentiment analysis dataset for better performance.

8. Model Optimization and Tuning

- **Hyperparameter Tuning with GridSearchCV**: Optimize model parameters for best performance.
  - from sklearn.model_selection import GridSearchCV; grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5); grid_search.fit(X_train, y_train)
- **Cross-Validation for Model Evaluation**: Use cross-validation to assess model robustness.

  - `from sklearn.model_selection import cross_val_score; scores = cross_val_score(model, X, y, cv=5)`
- **Using Early Stopping to Prevent Overfitting**: Implement early stopping during model training (applicable in deep learning models).
  - `from keras.callbacks import EarlyStopping; early_stopping = EarlyStopping(monitor='val_loss', patience=10); model.fit(X_train, y_train, validation_split=0.2, callbacks=[early_stopping])`
- **Ensemble Methods for Improved Performance**: Combine multiple models to improve prediction accuracy.
  - `from sklearn.ensemble import VotingClassifier; ensemble = VotingClassifier(estimators=[('model1', model1), ('model2', model2)], voting='soft'); ensemble.fit(X_train, y_train)`

9. Post-processing and Deployment

- **Serialize Model with joblib**: `joblib.dump(model, 'model_filename.pkl')`
- **Load a Model**: `model = joblib.load('model_filename.pkl')`
- **Create a Flask API for your model**: `@app.route('/predict', methods=['POST'])`
- **Use Docker to Containerize Your Application**: `FROM python:3.8-slim-buster`
- **Deploy to AWS Lambda using Serverless Framework**: `serverless deploy`
- **Use Streamlit for Quick Web Interfaces**: `streamlit run your_script.py`
- **Monitor Model Performance with MLflow**: `mlflow.log_metric("accuracy", accuracy)`
- **Update Model with Continuous Training**: `if performance_degrades: retrain_model()`
- **Use FastAPI for Modern Web APIs**: `@app.post("/predict/")`
- **Integrate Model with Messaging Platforms (Slack, Telegram) for Notifications**: `requests.post('SLACK_WEBHOOK_URL', json={'text': 'New prediction received'})`
- **Version Control for Models with DVC (Data Version Control)**: `dvc add data_directory`
- **Use TensorFlow Serving for Model Deployment**: `tensorflow_model_server --rest_api_port=8501 --model_name=sentiment_model --model_base_path="/path/to/model"`
- **Deploy to Google Cloud Run**: `gcloud run deploy --image gcr.io/project/image`
- **Use Azure ML for Deployment and Monitoring**: `az ml model deploy -n mymodel -m model:1`
- **Deploy to Heroku using Git**: `git push heroku master`

10. Libraries and Installation Commands

- **NLTK**: `!pip install nltk`
- **Scikit-learn**: `!pip install scikit-learn`
- **Pandas**: `!pip install pandas`
- **TextBlob**: `!pip install textblob`
- **spaCy**: `!pip install spacy`
- **gensim**: `!pip install gensim`
- **Beautiful Soup**: `!pip install beautifulsoup4`
- **Tweepy for Twitter data**: `!pip install tweepy`
- **Requests for HTTP requests**: `!pip install requests`
- **Hugging Face Transformers**: `!pip install transformers`
- **SHAP for model explanation**: `!pip install shap`
- **Matplotlib & Seaborn for visualization**: `!pip install matplotlib seaborn`
- **WordCloud for generating word clouds**: `!pip install wordcloud`

11. Application in Real-World Scenarios

- **Real-time Sentiment Analysis**: `stream = Stream(auth, listener); stream.filter(track=['keyword'])`
- **Sentiment Analysis in Multilingual Texts**: `translated = df['text'].apply(lambda x: GoogleTranslator(source='auto', target='en').translate(x)); df['sentiment'] = translated.apply(lambda x: TextBlob(x).sentiment.polarity)`
- **Aspect-based Sentiment Analysis**: `aspect_terms = extract_aspects(df['text']); df['aspect_sentiment'] = aspect_terms.apply(lambda x: TextBlob(x).sentiment.polarity)`
- **Sentiment Analysis Dashboard with Dash/Plotly**: `app = JupyterDash(__name__); app.layout = html.Div([...]); app.run_server(debug=True)`
- **Using Sentiment Analysis for Customer Feedback Analysis**: `feedback_sentiments = df['customer_feedback'].apply(lambda x: TextBlob(x).sentiment.polarity)`

12. Going Beyond Traditional Sentiment Analysis

- **Detecting Sarcasm in Texts**: `model_sarcasm = Sequential([...]); model_sarcasm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']);`
- **Emotion Detection**: `emotion_model = Sequential([...]); emotion_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']);`

- **Sentiment Analysis with Multi-Label Classification**: `model_multi_label = Sequential([...]); model_multi_label.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']);`
- **Integrating Sentiment Analysis into Chatbots**: `if detect_sentiment(message) < -0.5: respond_with_empathy(); else: respond_normally();`
- **Ethical Considerations and Bias Mitigation in Sentiment Analysis**: `bias_metrics = analyze_bias(sentiment_scores); if bias_metrics['unfairness'] > threshold: mitigate_bias();`

13. Leveraging Deep Learning and Transfer Learning

- **Using Pre-trained Word Embeddings (GloVe)**: `embeddings_index = dict(get_coefs(*o.strip().split()) for o in open('glove.6B.100d.txt')); embedding_matrix = np.zeros((len(word_index) + 1, 100)); for word, i in word_index.items(): embedding_vector = embeddings_index.get(word); if embedding_vector is not None: embedding_matrix[i] = embedding_vector`
- **Fine-tuning BERT for Sentiment Analysis**: `from transformers import BertTokenizer, TFBertForSequenceClassification; tokenizer = BertTokenizer.from_pretrained('bert-base-uncased'); model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased');`
- **Sentiment Analysis with Convolutional Neural Networks (CNNs)**: `model_cnn = Sequential([Embedding(input_dim=len(word_index)+1, output_dim=100, weights=[embedding_matrix], input_length=max_len, trainable=False), Conv1D(filters=128, kernel_size=5, activation='relu'), GlobalMaxPooling1D(), Dense(10, activation='relu'), Dense(1, activation='sigmoid')]); model_cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']);`
- **Sentiment Analysis with Recurrent Neural Networks (RNNs)**: `model_rnn = Sequential([Embedding(input_dim=len(word_index)+1, output_dim=100, weights=[embedding_matrix], input_length=max_len, trainable=False), SimpleRNN(128), Dense(10, activation='relu'), Dense(1, activation='sigmoid')]); model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']);`
- **Using LSTM for Sequence Modeling**: `model_lstm = Sequential(); model_lstm.add(Embedding(input_dim=len(word_index)+1, output_dim=100, weights=[embedding_matrix], input_length=max_len, trainable=False)); model_lstm.add(LSTM(128)); model_lstm.add(Dense(1, activation='sigmoid')); model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']);`