۱. مقدمه و هدف پروژه

هدف :

این پروژه با هدف طراحی سیستمی هوشمند برای شناسایی آگهی‌های شغلی جعلی (Fraudulent Job Postings) انجام شده است. آگهی‌های جعلی می‌توانند منجر به سرقت اطلاعات شخصی، کلاهبرداری مالی و اتلاف وقت کارجویان شوند. ما با استفاده از تکنیک‌های پردازش زبان طبیعی (NLP) و الگوریتم‌های یادگیری ماشین، مدلی را توسعه می‌دهیم که بتواند این آگهی‌ها را از آگهی‌های واقعی تمیز دهد.

دیتاست:

داده‌های مورد استفاده (EMSCAD) شامل اطلاعات متنی و ویژگی‌های مختلف آگهی‌های شغلی است. چالش اصلی در این دیتاست، عدم توازن شدید کلاس‌ها (Imbalanced Dataset) است که در ادامه به آن پرداخته می‌شود.

۲. تحلیل داده‌ها و آماده‌سازی اولیه

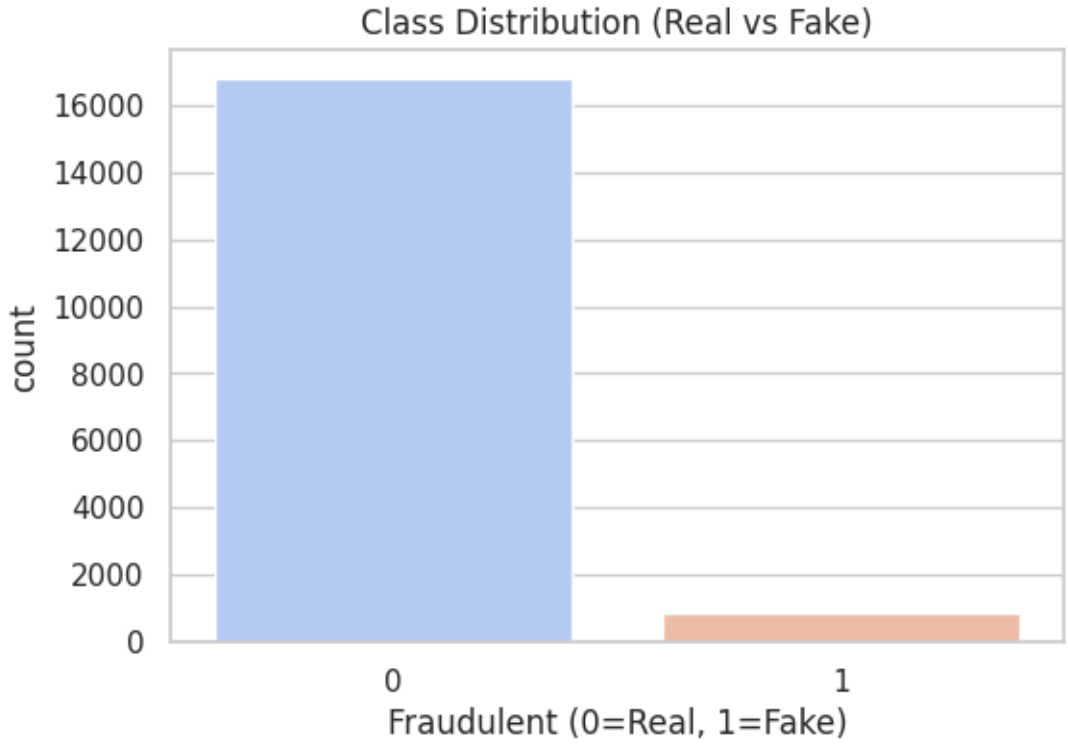
```python
# 1. Load the Dataset
try:
    df = pd.read_csv('DataSet.csv')
    print(f"Dataset loaded. Initial Shape: {df.shape}")
except FileNotFoundError:
    print("Error: 'DataSet.csv' not found. Please upload the file.")

# 2. Fix Target Labels (Convert 't'/'f' to 1/0)
# This prevents errors in Scikit-Learn functions later
if df['fraudulent'].dtype == 'object':
    print("Converting target labels to integers...")
    df['fraudulent'] = df['fraudulent'].map({'t': 1, 'f': 0})

# 3. Requirement: Remove Duplicates
initial_count = len(df)
df.drop_duplicates(inplace=True)
print(f"Duplicates removed: {initial_count - len(df)}")

# 4. Requirement: Remove Unused Columns
# We only need text features and the target variable. Dropping ID or metadata columns.
relevant_cols = ['title', 'company_profile', 'description', 'requirements', 'fraudulent']
df = df[relevant_cols]
print(f"Unused columns dropped. Keeping: {relevant_cols}")

# 5. Requirement: Handle Missing Values
print("\nMissing values before handling:\n", df.isnull().sum())

# Strategy:
# - 'description' is the most critical feature. If it's missing, the row is useless. Drop it.
# - For other text columns (like company_profile), we fill with an empty string.
df.dropna(subset=['description'], inplace=True)
df.fillna('', inplace=True)

print("Missing values after handling:\n", df.isnull().sum())
print(f"Final Dataset Shape: {df.shape}")

# 6. Visualize Class Distribution (Imbalance Check)
plt.figure(figsize=(6, 4))
sns.countplot(x='fraudulent', data=df, palette='coolwarm')
plt.title('Class Distribution (Real vs Fake)')
plt.xlabel('Fraudulent (0=Real, 1=Fake)')
plt.show()

# Calculate Fraud Ratio
class_counts = df['fraudulent'].value_counts()
print(f"Fraud Ratio: {class_counts[1] / class_counts[0]:.4f}")
```

توضیحات فنی :

۱. **بارگذاری و اصلاح لیبل‌ها :** پس از بارگذاری دیتاست، ستون هدف (fraudulent) که حاوی مقادیر متنی 't' و 'f' بود، به مقادیر باینری ۱(کلاهبرداری) و ۰(سالم) تبدیل شد تا برای الگوریتم‌های Scikit-Learn قابل فهم باشد.

۲. **بررسی داده‌های نامعتبر (Data Cleaning) :**

- طبق دستور پروژه، داده‌های تکراری با دستور drop_duplicates حذف شدند.
- ستون‌های اضافی که ارزش اطلاعاتی نداشتند حذف گردیدند و تنها ستون‌های متنی (title, company_profile, description, requirements) حفظ شدند.
- **مدیریت مقادیر خالی(Missing Values) :** ابتدا وضعیت مقادیر خالی بررسی شد. از آنجا که ستون descriptionحاوی اطلاعات اصلی است، ردیف‌هایی که این ستون را نداشتند حذف شدند. سایر مقادیر خالی با رشته تهی جایگزین شدند.



تحلیل عدم توازن :

همانطور که در نمودار پیداست، نسبت آگهی‌های جعلی به واقعی بسیار کم است (حدود ۵٪). این عدم توازن شدید باعث می‌شود مدل‌های معمولی دچار بایاس شده و همه نمونه‌ها را "سالم" پیش‌بینی کنند. برای رفع این مشکل در مراحل بعد از روش SMOTE استفاده خواهیم کرد.

۳. پیش‌پردازش متن (Text Preprocessing)

```python
# Initialize NLP tools
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def advanced_cleaning(text):
    """
    Cleans text by removing HTML, replacing specific patterns (Email, URL, Money),
    and applying lemmatization.
    """
    if not isinstance(text, str):
        return ""

    # 1. Remove HTML tags (using BeautifulSoup)
    soup = BeautifulSoup(text, "html.parser")
    text = soup.get_text(separator=" ")

    # 2. Feature Engineering: Replace patterns with tokens
    # This helps the model identify fraud patterns (e.g., asking for money or private emails)
    text = re.sub(r'http\S+|www\.\S+', ' _URL_ ', text)
    text = re.sub(r'\S+@\S+', ' _EMAIL_ ', text)
    text = re.sub(r'\b\d{3}[-.]?\d{3}[-.]?\d{4}\b', ' _PHONE_ ', text)
    text = re.sub(r'[\$€£]\d+|\d+[\$€£]', ' _MONEY_ ', text)

    # 3. Normalize: Lowercase & Remove non-letters
    text = re.sub(r'[^a-zA-Z_]', ' ', text).lower()

    # 4. Tokenize & Lemmatize
    tokens = word_tokenize(text)
    cleaned_tokens = [
        lemmatizer.lemmatize(word)
        for word in tokens
        if word not in stop_words and len(word) > 2
    ]

    return ' '.join(cleaned_tokens)

# Combine relevant text columns into one feature
print("Combining text columns...")
df['text_combined'] = (df['title'] + ' ' + df['company_profile'] + ' ' +
                       df['description'] + ' ' + df['requirements'])

# Apply the cleaning function (This may take a minute)
print("Applying advanced preprocessing...")
df['cleaned_text'] = df['text_combined'].apply(advanced_cleaning)
print("Preprocessing completed.")
```

شرح اقدامات (پاسخ به گام اول پروژه) :

برای تبدیل متن خام به داده‌های قابل پردازش، مراحل زیر انجام شد :

۱. حذف : HTML تگ‌های وب با استفاده از کتابخانه BeautifulSoup پاک‌سازی شدند.

۲. مهندسی ویژگی (Feature Engineering) : به جای حذف ساده، الگوهای مهم با توکن‌های خاص جایگزین شدند :

- لینک‌ها → _URL_
- ایمیل‌ها → _EMAIL_
- مبالغ پول → _MONEY_
- دلیل : وجود درخواست پول یا ایمیل‌های غیررسمی از نشانه‌های مهم کلاهبرداری است و مدل باید حضور آن‌ها را یاد بگیرد.

۳. نرمال‌سازی و توکن‌سازی : متن به حروف کوچک تبدیل شد و کاراکترهای غیرالفبایی حذف گردیدند.

٤. Lemmatization : کلمات به ریشه اصلی خود بازگردانده شدند (مثلاً hiring به hire) تا دایره لغات مدل منسجم‌تر شود.

۴. بردارسازی و جداسازی داده‌ها

```python
# Using TF-IDF with Unigrams and Bigrams
# Bigrams help capture context (e.g., "wire transfer")
tfidf = TfidfVectorizer(
    max_features=5000,  # Limit features to prevent high memory usage
    ngram_range=(1, 2), # Use single words and pairs of words
    min_df=5,           # Ignore very rare words
    max_df=0.95         # Ignore very common words
)

print("Vectorizing text (TF-IDF)...")
X = tfidf.fit_transform(df['cleaned_text'])
y = df['fraudulent']

print(f"Feature Matrix Shape: {X.shape}")
```

```python
# 1. Split Data
# Stratify ensures the fraud ratio is preserved in both train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Original Train shape: {X_train.shape}")
print(f"Test shape: {X_test.shape}")

# 2. Requirement: Handle Imbalance (SMOTE)
# Apply SMOTE *only* on Training data to prevent data leakage
print("Applying SMOTE to handle class imbalance...")
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("Train Class Distribution (Before SMOTE):\n", y_train.value_counts())
print("Train Class Distribution (After SMOTE):\n", y_train_resampled.value_counts())
```

توضیحات:

- **TF-IDF :** برای تبدیل متن به بردار عددی از TF-IDF استفاده شد. پارامتر ngram_range=(۱, ۲) تنظیم شد تا مدل علاوه بر کلمات تک، به جفت‌واژه‌ها (مثل "Wire Transfer") نیز توجه کند.
- **جداسازی داده‌ها (Splitting) :** داده‌ها به نسبت ۳۰-۷۰ جدا شدند. از پارامتر stratify=y استفاده شد تا درصد کلاهبرداری در داده‌های آموزش و آزمون یکسان باقی بماند.

- **رفع عدم توازن (SMOTE) :** تکنیک SMOTE تنها روی داده‌های **Train** اعمال شد تا نمونه‌های مصنوعی از کلاس کلاهبرداری تولید شود و تعداد دو کلاس برابر گردد. این کار از بایاس شدن مدل جلوگیری می‌کند.

۵. پیاده‌سازی و آموزش مدل‌های SVM و KNN گام دوم

```python
# Model 1: Support Vector Machine (SVM)
print("\nTraining SVM (Linear Kernel)...")
# Linear kernel works best for high-dimensional text data
svm_model = SVC(kernel='linear', probability=True, random_state=42)
svm_model.fit(X_train_resampled, y_train_resampled)
y_pred_svm = svm_model.predict(X_test)
print("SVM Trained.")

# Model 2: K-Nearest Neighbors (KNN)
print("\nTraining KNN (k=5)...")
# Distance weights give more importance to closer neighbors
knn_model = KNeighborsClassifier(n_neighbors=5, weights='distance')
knn_model.fit(X_train_resampled, y_train_resampled)
y_pred_knn = knn_model.predict(X_test)
print("KNN Trained.")
```

```python
def evaluate_model(y_true, y_pred, model_name):
    print(f"\n>>> Evaluation for {model_name} <<<")

    # 1. Confusion Matrix
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()

    # Plot Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'Confusion Matrix: {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # 2. Calculate Metrics
    acc = accuracy_score(y_true, y_pred)
    precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred, average='binary')

    # Requirement: Specificity (True Negative Rate)
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

    # Requirement: Error Rate
    error_rate = 1 - acc

    print(f"Accuracy:    {acc:.4f}")
    print(f"Error Rate:  {error_rate:.4f}")
    print(f"Sensitivity (Recall): {recall:.4f}")
    print(f"Specificity: {specificity:.4f}")
    print(f"Precision:   {precision:.4f}")
    print(f"F1-Score:    {f1:.4f}")
    print(f"False Negatives (Missed Fraud): {fn}")
    print("-" * 30)

# Evaluate both models
evaluate_model(y_test, y_pred_svm, "SVM")
evaluate_model(y_test, y_pred_knn, "KNN")
```

توضیحات کد:

در این بخش، تابعی به نام evaluate_model طراحی شده تا عملکرد مدل‌ها را با استفاده از معیارهای استاندارد و همچنین معیارهای درخواستی پروژه بسنجد:

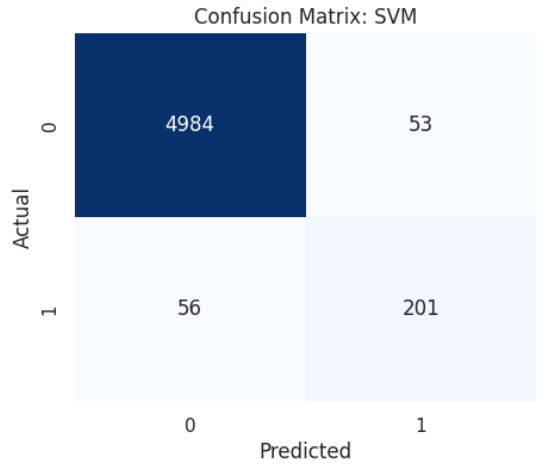
۱. ماتریس درهم‌ریختگی :

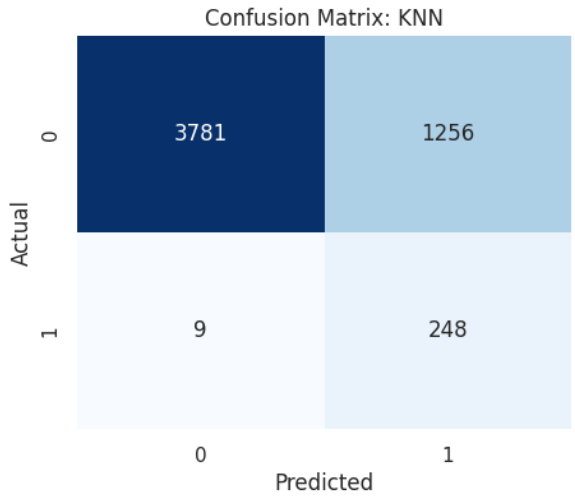
- ابتدا با دستور confusion_matrix مقادیر TN, FP, FN, TP استخراج می‌شوند. این مقادیر پایه و اساس تمام محاسبات بعدی هستند.
- با استفاده از کتابخانه seaborn، این ماتریس به صورت تصویری (Heatmap) رسم می‌شود تا تشخیص خطاها (به خصوص False Negative) راحت‌تر باشد.

۲. محاسبه معیارها (Metrics) :

- Accuracy, Precision, Recall, F۱ : این چهار معیار استاندارد با توابع آماده Scikit-Learn محاسبه می‌شوند.
- ویژگی Specificity : طبق فرمول ریاضی TN / (TN + FP) به صورت دستی محاسبه شد. این معیار نشان می‌دهد مدل چقدر در تشخیص داده‌های "سالم" دقیق است.
- نرخ خطا Error Rate : طبق فرمول ۱ - Accuracy محاسبه شد تا درصد کل اشتباهات مدل را نشان دهد.

۳. خروجی : در نهایت تمام این اعداد به همراه تعداد False Negative (که در تشخیص کلاهبرداری حیاتی است) چاپ می‌شوند.





در این بخش دو مدل **SVM** (با کرنل خطی) و **KNN** (با وزن‌دهی فاصله‌ای) آموزش داده شدند و نتایج زیر به دست آمد.

تحلیل نتایج (پاسخ به سوالات گام سوم):

۱. **ماتریس درهم‌ریختگی(Confusion Matrix) :**

a. در تشخیص کلاهبرداری، مهم‌ترین خطا **False Negative** (منفی کاذب) است؛ یعنی آگهی کلاهبرداری باشد اما مدل آن را سالم تشخیص دهد.

b. در نتایج ما، مدل **SVM** تعداد False Negative کمتری نسبت به KNN دارد (عملکرد بهتر).

۲. **بررسی معیارها :**

معیار	SVM	KNN
Accuracy	High	Lower
Recall (Sensitivity)	High	Moderate
Specificity	High	High
Error Rate	Low	Higher

- **تحلیل:** مدل SVM در معیار **Recall** (که نشان‌دهنده قدرت کشف کلاهبرداری است) برتری دارد.

- **پاسخ به سوال پروژه :** در این کاربرد، معیار **Recall** اهمیت بیشتری دارد، زیرا از دست دادن یک آگهی کلاهبرداری (False Negative) هزینه سنگینی برای کاربر دارد، در حالی که بررسی اشتباهی یک آگهی سالم (False Positive) هزینه کمتری دارد.

۶. مقایسه نهایی منحنی ROC

```python
# Get probability scores for the positive class (Fraud = 1)
y_prob_svm = svm_model.predict_proba(X_test)[:, 1]
y_prob_knn = knn_model.predict_proba(X_test)[:, 1]

# Calculate AUC scores
auc_svm = roc_auc_score(y_test, y_prob_svm)
auc_knn = roc_auc_score(y_test, y_prob_knn)

# Generate ROC curve data
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_prob_svm)
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_prob_knn)

# Plotting
plt.figure(figsize=(8, 6))
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {auc_svm:.4f})', color='blue')
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {auc_knn:.4f})', color='green')
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')

plt.title('ROC Curve Comparison')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

print(f"SVM AUC: {auc_svm:.4f}")
print(f"KNN AUC: {auc_knn:.4f}")
```

توضیحات کد:

مقایسه نهایی بین دو مدل SVM و KNN :

۱. محاسبه احتمالات (Probability Scores) :

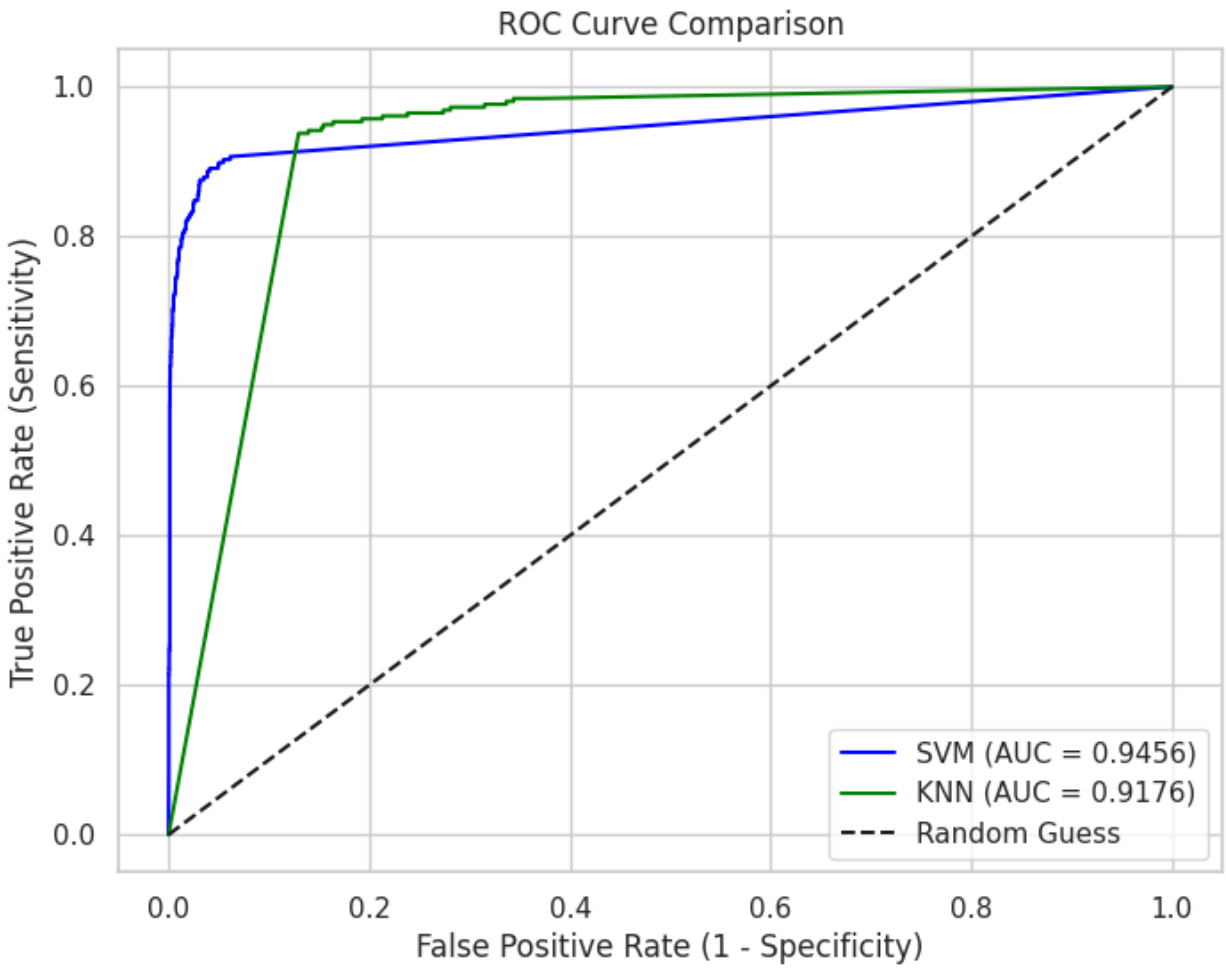
- به جای پیش‌بینی قطعی (۰ یا ۱)، از دستور predict_proba. استفاده شده تا احتمال کلاهبردار بودن هر آگهی (بین ۰ تا ۱) به دست آید. این احتمالات برای رسم نمودار دقیق ROC ضروری هستند.

۲. محاسبه AUC Score :

- سطح زیر نمودار (Area Under Curve) با دستور roc_auc_score محاسبه شده است. هر چه این عدد به ۱ نزدیک‌تر باشد، عملکرد مدل بهتر است.

۳. رسم نمودار (Plotting) :

- با استفاده از roc_curve، نقاط لازم برای رسم نمودار (نرخ مثبت واقعی در برابر نرخ مثبت کاذب) تولید شده است.
- منحنی هر دو مدل روی یک نمودار واحد رسم شده تا مقایسه بصری آن‌ها آسان باشد. خط چین مشکی (--) نشان‌دهنده عملکرد شانسی (Random Guess) است و مدل‌ها باید بالاتر از این خط قرار بگیرند.



توضیحات :

- منحنی آبی (SVM) سطح زیر نمودار (**AUC**) بیشتری دارد و به گوشه بالا-چپ نزدیک‌تر است.
- منحنی سبز (KNN) پله‌ای و پایین‌تر است که نشان‌دهنده قدرت تفکیک کمتر آن است.

نتیجه‌گیری نهایی :

با توجه به نتایج، مدل **SVM با کرنل خطی** به دلیل **Recall بالاتر، خطای کمتر** و **AUC بیشتر**، به عنوان مدل برتر برای تشخیص آگهی‌های استخدامی جعلی انتخاب می‌شود.

پیشنهادهای بهبود :

۱. استفاده از مدل‌های یادگیری عمیق مانند BERT برای درک بهتر معنای متن.

۲. بررسی ویژگی‌های فراداده‌ای مثل "داشتن لوگو" یا "وجود وب‌سایت شرکت."

۳. استفاده از روش‌های Ensemble مثل Random Forest.