wt_petase_model_creation

February 12, 2025

1 Creating a competative supervised model for wild type PETase activity at low pH

We show in a seperate paper TODO that supervised models should be used for predicting PETase activity at unique conditions (eg. low pH) once some assay labeled data is available, and this outperforms HMMs. Here we create models that: 1. Take in embeddings as input, explore over: Aligned OHE, ES|M2, SaProt, MSATransformer 2. Use linear vs non-linear models: Linear regression, Random Forest

Hyperperameter optimization is conducted over the models for each input type.

Save the final model, which can be loaded like any other sklearn model if AIDE is installed.

eg. `model=joblib.load('model.pkl')`

[ ]:
```
import os

import pandas as pd
import numpy as np
from sklearn.model_selection import RandomizedSearchCV, KFold, cross_validate
from sklearn.feature_selection import VarianceThreshold
from sklearn.linear_model import ElasticNet, LinearRegression
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestRegressor
from scipy.stats import loguniform, spearmanr
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("white")
sns.set_context("talk")

import aide_predict as ap
from aide_predict.utils.data_structures.structures import StructureMapper
```

1.1 1. Load and prepare data

We need to get: 1. The sequences and labels 2. Assign their structures (for SaProt Embedding) 3. Get and MSA of known PETases (for baseline HMMscore and MSA transformer)

[ ]:
```
RAW_DATA_DIR = os.path.join('.', 'data', 'p740')
```

1.1.1 1.1 Label data

[ ]:
```
df = pd.read_csv(os.path.join(RAW_DATA_DIR, 'label_data.csv'), index_col=0).
↪sample(frac=1, random_state=42)
```

[ ]:
```
df = df.dropna(subset=['activity_at_5.5_40_cryPow'])
# drop rows with non canonical AAs - these were not predicted properly by AF
X = ap.ProteinSequences.from_dict(df['sequence'].to_dict())
has_non_canonical = [x.has_non_canonical for x in X]
df = df[~np.array(has_non_canonical)]
```

[ ]:
```
X = ap.ProteinSequences.from_dict(df['sequence'].to_dict())
y = df['activity_at_5.5_40_cryPow'].values
```

[ ]:
```
sns.kdeplot(y, bw_adjust=0.5)
plt.xlabel('Activity at 5.5 pH, 40°C')
```

1.1.2 1.2 Structures

[ ]:
```
# get the structures - Needed for SaProt embedding
mapper = StructureMapper(os.path.join(RAW_DATA_DIR, 'structures'))
mapper.assign_structures(X)
```

1.1.3 1.3 Homolog MSA (for MSA transformer)

Compute weights so that MSA transformer can sample it properly.

[ ]:
```
msa = ap.ProteinSequences.from_fasta(os.path.join(RAW_DATA_DIR, 'D1-Scraped-513.
↪mfa'))
msa.aligned
```

[ ]:
```
msa.width
```

1.2 2. Define scoring functions

[ ]:
```
# 5 fold cv
cv_obj = KFold(n_splits=5, shuffle=True, random_state=42)
```

[ ]:
```
# metrics to measure
# marks magnitude of error eg R2, also score AUROC to see if the model can␣
↪classify active or not
scoring = {
```

```
    'spearman': lambda est, X, y: spearmanr(y, est.predict(X))[0],
    'roc_auc': lambda est, X, y: roc_auc_score(y > 0.001, est.predict(X))
}
```

[ ]:
```
def construct_pipeline(embedder, model, pca: bool=False):
    if not pca:
        return Pipeline([
            ('embedder', embedder),
            ('var', VarianceThreshold()),
            ('scaler', StandardScaler()),
            ('model', model)
        ])
    else:
        return Pipeline([
            ('embedder', embedder),
            ('var', VarianceThreshold()),
            ('scaler', StandardScaler()),
            ('pca', PCA(n_components=0.95)),
            ('model', model)
        ])

def evaluate_pipeline_with_hyperopt(embedder, model_info, n_iter=10):
    """Run hyperparameter optimization on a pipeline with a given embedder and␣
 ↪model

    Params:
    embedder: Embedder object eg ap.BaseProteinModel
    model_info: dict with keys:
        'model': sklearn model object
        'param_distributions': dict of hyperparameter distributions for␣
 ↪RandomizedSearchCV
    """
    pipeline = construct_pipeline(embedder, model_info['model'])
    random_search = RandomizedSearchCV(
        pipeline,
        param_distributions=model_info['param_dist'],
        n_iter=n_iter,
        cv=cv_obj,
        scoring=scoring,
        refit='spearman',
        n_jobs=1)
    random_search.fit(X, y)

    best_params = random_search.best_params_
    cv_scores = cross_validate(pipeline.set_params(**best_params), X, y,␣
 ↪cv=cv_obj, scoring=scoring)
    return best_params, cv_scores
```

1.3 3. Baseline model: HMM

[ ]:
```
hmm = ap.HMMWrapper()
hmm.fit(msa)

baseline_scores = {
    k: v(hmm, X, y) for k, v in scoring.items()
}
print('Baseline scores:', baseline_scores)
```

1.4 4. Supervised learning: Define embedders, models, and hyperparameter space

[ ]:
```
embedders = {
    'ESM2': ap.ESM2Embedding(
        metadata_folder='esm2_embeddings',␣
 ↪model_checkpoint='esm2_t33_650M_UR50D', device='mps', pool='mean'),
    'SaProt': ap.SaProtEmbedding(metadata_folder='saprot_embeddings',␣
 ↪device='mps', pool='mean'),
    'MSATransformer': ap.MSATransformerEmbedding(
        metadata_folder='msa_embeddings', device='mps', pool='mean',
        n_msa_seqs=32
    ),
    'AlignedOneHot': ap.OneHotAlignedEmbedding(
        metadata_folder='onehot_embeddings')
}
# fit the models that have fixed fitting over folds
embedders['ESM2'].fit([])
embedders['SaProt'].fit([])
embedders['MSATransformer'].fit(msa)
```

[ ]:
```
models = {
    'ElasticNet': {
        'model': ElasticNet(),
        'param_dist': {
            'model__alpha': loguniform(1e-5, 1e2),
            'model__l1_ratio': np.linspace(0, 1, 11)
        }
    },
    'RandomForest': {
        'model': RandomForestRegressor(n_estimators=10),
        'param_dist': {
            'model__max_depth': [None, 10, 100],
            'model__min_samples_split': [2, 5, 10],
            'model__min_samples_leaf': [1, 5, 10]
        }
    }
```

```
}
```

1.5 5. Train and evaluate models with hyperparameter optimization

[ ]:
```
import joblib
if not os.path.exists('search_results.pkl'):
    results = {}
else:
    results = joblib.load('search_results.pkl')
```

[ ]:
```
for embedder_name, embedder in embedders.items():
    for model_name, model_info in models.items():
        if f'{embedder_name}_{model_name}' in results:
            print(f"Skipping {embedder_name} with {model_name}...")
            continue
        else:
            print(f"Evaluating {embedder_name} with {model_name}...")
        best_params, scores = evaluate_pipeline_with_hyperopt(embedder,␣
 ↪model_info, n_iter=200)
        results[f'{embedder_name}_{model_name}'] = {
            'embedder': embedder_name,
            'model': model_name,
            'best_params': best_params,
            'spearman': scores['test_spearman'],
            'roc_auc': scores['test_roc_auc']
        }
        joblib.dump(results, 'search_results.pkl')
```

[ ]:
```
# convert to long
df_list = []
for item in results.values():
    for i in range(5):  # Assuming 5 values for each metric
        df_list.append({
            'embedder': item['embedder'],
            'model': item['model'],
            'spearman': item['spearman'][i],
            'roc_auc': item['roc_auc'][i]
        })

df = pd.DataFrame(df_list)

# Melt the DataFrame to create a column for the metric type
df_melted = pd.melt(df, id_vars=['embedder', 'model'], var_name='metric',␣
 ↪value_name='value')
```

[ ]:
```
df_melted.to_csv('fig3_data.csv', index=False)
```

[ ]:
```
plt.figure(figsize=(8, 10))

# Create the faceted plot
g = sns.catplot(
    data=df_melted,
    kind="bar",
    x="model",
    y="value",
    hue="metric",
    col="embedder",
    height=4,
    aspect=1.0,
    palette="Set2",
    col_wrap=4,
    ci="sd",
    legend=True,  # We'll add the legend manually
    # change color
    edgecolor='black',
    linewidth=2

)
# remove legend from seaborn
g._legend.remove()
# add baselines of the same color as the bars
for ax in g.axes.flat:
    for i, metric in enumerate(['spearman', 'roc_auc']):
        ax.axhline(baseline_scores[metric], color=sns.color_palette("Set2")[i],␣
 ↪linestyle='--', label=f'HMM (baseline) {metric}')
    ax.set_ylim(0, 1)

plt.legend(loc='upper center', bbox_to_anchor=(-1.1, -0.18), ncol=4)
# Customize the plot
g.set_axis_labels("")
g.set_titles("{col_name}")

# Display the plot
plt.savefig('p740_model_comparison.png', bbox_inches='tight', dpi=300)
```

1.6 6. Train final model and save

[ ]:
```
best_row = df.groupby(['embedder', 'model']).mean().sort_values('roc_auc',␣
 ↪ascending=False).iloc[0]
best_row
```

[ ]:
```
best_pipeline = construct_pipeline(embedders[best_row.name[0]], models[best_row.
 ↪name[1]]['model'])
```

[ ]:
```
best_params = results[f'{best_row.name[0]}_{best_row.name[1]}']['best_params']
```

[ ]:
```
models[best_row.name[1]]['model']
```

[ ]:
```
best_pipeline.set_params(**best_params)
```

1.7 First do a CV prediction so we can plot parity

[ ]:
```
y_trues = []
y_preds = []
for train_idx, test_idx in cv_obj.split(X):
    best_pipeline.fit(X[train_idx], y[train_idx])
    y_pred = best_pipeline.predict(X[test_idx])
    y_trues.append(y[test_idx])
    y_preds.append(y_pred)

y_trues = np.concatenate(y_trues)
y_preds = np.concatenate(y_preds)
y_trues = y_trues > 0.001
```

[ ]:
```
y_preds.shape
```

[ ]:
```
df_
```

[ ]:
```
def Find_Optimal_Cutoff(target, predicted):
    """ Find the optimal probability cutoff point for a classification model␣
 ↪related to event rate
    Parameters
    ----------
    target : Matrix with dependent or target data, where rows are observations

    predicted : Matrix with predicted data, where rows are observations

    Returns
    -------
    list type, with optimal cutoff value

    """
    from sklearn.metrics import roc_curve
    fpr, tpr, threshold = roc_curve(target, predicted)
    i = np.arange(len(tpr))
    roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i), 'threshold' :␣
 ↪pd.Series(threshold, index=i)})
    roc_t = roc.iloc[(roc.tf-0).abs().argsort()[:1]]

    return list(roc_t['threshold'])
```

```
cutoff = Find_Optimal_Cutoff(y_trues, y_preds)
cutoff
```

[ ]:
```
precision = np.sum((y_preds > cutoff) & y_trues) / np.sum(y_preds > cutoff)
precision
```

[ ]:
```
fig, ax = plt.subplots(figsize=(8, 2))

df_ = pd.DataFrame({
    'y_true': y_trues,
    'y_pred': y_preds,
})

sns.boxplot(data=df_, x='y_pred', y='y_true', ax=ax, orient='h')
ax.set_xlabel('Model score')
ax.set_ylabel('Experimentally active \nat pH=5.5, 40°C')
ax.set_title('Precision (expected hit rate): {:.2f}'.format(precision))

ax.fill_between([ax.get_xlim()[0], cutoff[0]], [ax.get_ylim()[0], ax.
 ↪get_ylim()[0]], [ax.get_ylim()[1], ax.get_ylim()[1]], color='grey', alpha=0.
 ↪5)
ax.vlines(cutoff[0], ax.get_ylim()[0], ax.get_ylim()[1], color='red',␣
 ↪linestyle='--', label='Decision boundary')
plt.legend(bbox_to_anchor=(.25, -.4,), loc='upper left')
plt.savefig('p740_precision.png', bbox_inches='tight', dpi=300)
```

[ ]:
```
best_pipeline = best_pipeline.fit(X, y)
```

[ ]:
```
preds = best_pipeline.predict(X)
```

[ ]:
```
joblib.dump(best_pipeline, 'p740_best_pipeline.pkl')
```

1.8 7. Load model and predict

[ ]:
```
import joblib
best_pipeline = joblib.load('p740_best_pipeline.pkl')
```

[ ]:
```
preds = best_pipeline.predict(X)
```

[ ]:
```
preds
```

[ ]:
```

```